Docket No. 0142-0362P
(PATENT)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
GARDEREN VAN, F. et al.

Application No.: 09/974,911 Confirmation No.: 8931

Filed: October 12, 2001 Art Unit: 2127

For: DISTRIBUTED DOCUMENT HANDLING SYSTEM Examiner: K. Tang

PROPOSED DECLARATION UNDER 37 C.F.R. §1.131

Assistant Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

I, Jack van de Sande, residing at **Schoolweg 24, 5916 PL, Venlo**, The Netherlands do declare and say as follows:

1. I am a citizen and a qualified patent attorney under the laws of The Netherlands. I am also a European Patent Attorney.

2. I was responsible for drafting Application No. 00203538.4, which was filed in Europe on October 13, 2000.

3. I understand that the Examiner in charge of the above-identified application has made a rejection under 35 U.S.C. § 102(e) as being anticipated by Whitmarsh et al., U.S. Patent No. 7,042,585 (hereinafter "the Whitmarsh et al. Patent").

4. The Whitmarsh et al. Patent issued on May 9, 2006, which is after the U.S. filing date of the present application of October 12, 2001. Therefore, the Examiner is relying on the

filing date of October 10, 2000 of Whitmarsh et al. as the reference date of Whitmarsh et al. (35 U.S.C. § 102(e) date). Accordingly, the effective date of the Whitmarsh et al. reference is **October 10, 2000**.

5. A verified translation of the European priority application of the present application, i.e. European Application No. 00203538.4, is not necessary to perfect the claim to foreign priority of the present application, because the European priority application was filed in the English language on October 13, 2000. A certified copy of the European priority application was submitted to the U.S. Patent Office on October 13, 2000. Therefore, the effective filing date of the present application is October 13, 2000.

6. The present invention was conceived **prior to October 10, 2000** and the present invention was constructively reduced to practice on October 13, 2000 by diligently filing European Application No. 00203538.4 on October 13, 2000.

7. The present invention was conceived and reduced to practice in The Netherlands, which is a WTO member country.

8. As evidence of prior invention, the following facts and documents are provided:

(A). A draft application of European Application No. 00203538.4, which was prepared prior to October 10, 2000. It is noted that the date on the draft application has been redacted; however, I submit that the draft application was prepared prior to October 10, 2000 and is therefore evidence of conception of the present application prior to the effective date of the Whitmarsh et al. reference of October 10, 2000.

(B). A draft set of drawings for European Application No. 00203538.4, which were prepared prior to October 10, 2000.

(C). The European Application was then diligently filed on October 13, 2000.

(D) In view of the above, the Whitmarsh et al. Patent is not available as a reference against the present application.

9. I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and such willful false statements may jeopardize the validity of the application or any patent issuing thereon.

May 15, 2008

Date

By _____

Jack van de Sande

Enclosure: Draft Specification and drawings

Distributed document handling system

Description

FIELD OF THE INVENTION

The invention relates to a distributed document handling system for carrying out jobs, where jobs are carried out by services distributed over a network and where a job leads to a product as a result from a production process, comprising: a pool of services the services being distributed over a number of interconnected processing devices; means for entering a job specification comprising product specifications specifying the product to be delivered by the job; and means for determining a path of services, the services being selected from the pool of services, suitable to carry out the job in accordance with the product specifcations.
The term path of services is used to denote the set of services that are involved in realizing the requested job and which are not necessarily a set of services that are executed in sequence but may also encompass services that have to be executed in parallel.

BACKGROUND OF THE INVENTION

With the advent of digital technology the digital copier became feasible. Basically a digital copier comprises a scanner for converting hard copy images into electronic images and a printer for converting electronic images into hard copy images. Between scanner and printer the image is available as an electronic digital image. It is this characteristic that provides the ability to provide a digital copier with a wealth of features that were not feasible before. Due to the digital technology the copier was enhanced by all kind of features that became possible now, such as queueing of jobs, electronic rdf, image enhancements by performing digital signal processing on the electronic images, editing of images, pattern recognition processing etc.

Furthermore these apparatus were able to communicate the electronic images, so that such a copier could be used as a printer to print images received from a host computer and could be used as a fax apparatus to exchange images with other fax machines.

Recently it became also known to use the communication facility to exchange print job images from one reproduction apparatus with another reproduction apparatus with the purpose to process the images on the other apparatus, where after the processed images are sent back to the originating reproduction apparatus and printed out.

Such a facility is described in EPA 0797344 of Sharp

Sharp describes a distributed document handling system comprising a plurality of image forming apparatuses that are operatively connected so as to communicate with each other. Each image forming apparatus comprises a number of services that perform image processing functions. Disclosed in Sharp is a system wherein the aparatuses can exchange image information with each other through the communication apparatus. (b)

As shown in the system described above in a dynamic distributed environment tasks can be performed by linking services that carry out subtasks.

However, the more services are available in such an environment, the more combinations of services are possible. Furthermore as the number of available services increases, one may expect the number of combinations that perform a desired task to rise as well. In such case a system according to the prior art will become unmanageable. Furthermore not all such combinations will be desirable in the light of restrictions that are imposed by the environment or by the requestor. The prior art does not address these problems.

SUMMARY OF THE INVENTION

The object of the invention is to obviate the above-described disadvantages and to provide a distributed information processing system that efficiently and effectively support the linking of services and yields combinations of services that fullfil t he needs of a user to a greater extent.

To this end the distributed information processing system according to the preamble is improved such that the job specification also comprises specifications specifying circumstantial constraints without effect on the product and such that the means for determining the path of services take into account circumstantial constraints for that job.

This takes away the disadvantage that the number of combinations becomes unmanageable high. In selecting a path the system according to the invention is also able to take into account restrictions regarding e.g. price, reliability and secrecy. The measure according to the invention allows the user to apply more constraints, also constraints not reflected in the final product to be deliverd by the job, so that the number of solutions provided by the system is more in compliance with the needs of the user.

In a further embodiment a circumstantial constraint defines a limit in an ordered range; and the system also comprises means for ranking paths suitable to carry out the job in accordance with the ordered range of the circumstantial constraint and user interface means for selection by the user of a desired job specification from a ranked list of job specifications based on the ranked paths. A circumstantial constraint may be such that it is expressed as a value range in a certain domain, e.g. the price has to be lower than a certain amount expressed in a certain currency.

This allows the generation of a number of paths that all fullfill the constraints however with variations in the values found for a certain circumstantial constraint. In this way optimization is possible for a certain constraint.

In a further embodiment the system comprises user interface means for selection by the user of the circumstantial constraint to be used in the ranking of the paths. Most of the time a number of paths suited to carry out the job will be delivered by the system. These paths will differ in the extent to which the conditional constraints are fulfilled. This feature according to the invention provides the user with a faciltiy to easily grasp the path that best suits his needs.

In yet a furthter embodiment the circumstantial constraint is a total price of the job to be carried out and the system comnprises means for calculating the total price from price attributes of services comprised in a determined path. In this way a user can easily find the cheapest way for the carrying out of the job.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be explained in detail with reference to the accompanying drawings wherein

Fig.1 Distributed system

Fig.2 Architecture of interconnected peripherals

Fig.3 Records of application services

Fig.4 Example of a workstation provided with service management components

Fig.5 Example of a di graph of application services

Fig.6 Flow diagram job submission handler

Fig.7 Objects used in the system

Fig.8 Operating screen of the job submission handler

Fig.9 Flow diagram getpath method.

DESCRIPTION OF THE EMBODIMENTS

Fig.1 diagrammatically illustrates an embodiment of a distributed document handling system according to the invention. The image forming system 101 comprises a number of processing devices or peripherals 102 – 110 interconnected via a network 111 and thus able to communicate with each other. In particular by way of example peripheral 110 is a low volume copier, peripheral 108 is a scanner, peripheral 104 is a high volume printer, peripheral 107 is a file server, peripheral 105 is a general server, peripherals 102, 103, 106 and 109 are workstations.

The distributed document handling system according to the invention carries out jobs on documents. Documents may be described or characterised in a variety of ways. For the description of the invention a document will be characterised in dependence of actual properties a document has at a certain moment. Such properties determine a state the document is in at that moment. Such a state corresponds with a particular manifestation of the document at a certain moment. In a first state the document may be manifested as a particular file encoded in ascii, in a next state the document may be manifested as a hard copy, printed on A4-sheets, double sided and stapled. A job will be expressed now as a transformation of a document with an initial state s_i to the document with a target state s_t.

According to the invention a job is carried out by services distributed over the netwrk. These services are available at various peripherals in the network and perform specific functions on documents. Examples of such functions are image processing, printing, scanning, encrypting and conversion from one page description language to another page description language. These kind of services will be referred to further as application services. Typical a number of application services will be involved in carrying out a job. The effect of each of these application services, when processing a document, will also be described with reference to the change of state of the document. Starting from an initial state application services selected according to the invention will process the document in sequence and/or parallel whereby after each processing the document will change state, until the target state s_t has been reached. The application services involved are denoted in a digraph that determines at which moment in the processing of a job an application service is next. Fig. 5 shows an example of such a digraph of application services (solid black). A node in the digraph represents a particular state a document is in and an edge in the digraph represents an application service that brings the document from a first state to a second state. In the digraph of services shown the document is transformed stepwise from its intial state via a number of intermediate states to the target state, where each step is accomplished by an application service. In the description a state will be presented as an enumeration of explicitly typed parameter values.

COMPUTATIONAL ENVIRONMENT

Fig.2 illustrates the global architecture of an interconnected peripheral 201. Each peripheral has its own operating system 202. Such an operating system performs basic tasks and defines the environment in which application programs will be run. Each peripheral has a number of application programs 203 that carry out specific tasks. Peripherals are connected with each other via a network link 205 and thus a networked system is formed. Via the network links it is possible that application programs on different peripherals interact with each other.

In a particular implementation some application programs are programmed in the Java language and need a Java Virtual Machine to be able to run in the peripheral. The Java Virtual Machine is provided as a layer 204 above the particular OS. As a consequence all these application programs are able to run on all the peripherals despite the underlying different operating systems.To let the whole set of interconnected heterogeneous peripherals behave as one virtual machine from the viewpoint of the application a middleware layer 206 is provided. A middleware layer well known in the art is e.g. JINI of Sun MicroSystems. However, any other environment that offer applications, running on a variety of heterogeneous computers, means for interacting with each other is equally applicable.

According to the invention at various peripherals in the network application services are available that perform specific functions on documents. All the application services together, distributed over the various peripherals in the system, make up a pool of services. For the purpose of the management of these application services the middleware layer is extended with a service management system.

The service management system will further be explained with reference to Fig. 4. Fig. 4 depicts an embodiment of a workstation 401 provided with a service management system 402 as part of the middleware layer 403, a number of application services 404 and local applications 405 and 406.

The service management system comprises a number of service management components. Basic service management components are available on each node all the time. Other service management components will be locally available as needed. The service management system takes care that the application services can be addressed both locally and remotely. Service management components can only be addressed by application services locally. If needed a proxy is made for local access. The plurality of service management components comprises a registry 406, a path composition module 407, a transaction manager 408 and a mobility manager 409.

For the description of the system according to the invention the object-oriented model will be used. This means that where appropriate the invention will be described with reference to classes and objects instantiated from these classes. An object can be envisaged as a container of data and this data can be accessed only via methods provided by the object. A persistant object is an object that is saved and restored automatically upon exiting and starting of an application. Instantiation of an object from a class will be indicated by the keyword "new". Background information regarding the java language and object oriented programming can be found in the publications "The Java™ Platform, A White Paper" by Douglas Kramer and xxxxx.

The path composition module 407 according to the invention determines zero or more paths of application services that fulfil a job request. In this respect a job request is referred to as a request object instantiated from the class Request.The class Request has as data members: Inputstate, OutputState, and Constraints. Methods supported are getInput(), getOutput() and getConstraints(). Typical most application services will perform only very specific tasks and as a consequence in most cases the pool of services will not contain one service that exactly matches with the request of the user. Thus most of the time the request can be matched only by a combination of application services. However not every combination is desirable. Some paths may contain slow services or may contain expensive services. It is in particular the task of the path composition system according to the invention to find a suitable combination of

services, operating within constraints specified by the user. The path composition module is implemented as the object PathComposer. The object provides the method PathComposer.getPath(Request) that will return zero or more paths that are in compliance with Request.

The registry 406 registers and deregisters application services and retrieves application services according to requested attributes and returns an identifier for invocation of the service. For this purpose the registry internally manages a database of available application services. The database contains records of application services as depicted in Fig. 3. All data of an application service are stored in an application service record. Each application service is described by a type, a global unique identifier (GUID) and a set of attributes. The type gives a generic indication of the function the service carries out. Attributes define the characteristics of a service: e.g. the specific function it carries out, physical characteristics like location, and accounting characteristics like price per print or price per unit of data. From appropriate attributes of a service all possible transformations from a first state to second state that a service is able to carry out can be extracted e.g. by parsing means known in the art. An accounting scheme may be defined by reference to one of a number of schemes, or it is defined explicitly as shown in the examples of Fig. 3. Fig. 3 will be detailed discussed in the elaboration of the examples.

The registry also contains a table for mapping the GUID on an adress in the adress space of the networked system. Similarly the registry also manages a database of service management components that are distributed over the network, like e.g. the path composition component.

The registry is implemented as the object Registry. Methods available for access of data are getService(), getAttributes() and getOutputStates(). The method Registry.getService(retrieval_criterion) retrieves a service from the registry. This method returns an identifier of a service that fulfils the retrieval criterion. The method Registry.getAttributes(Service) returns attributes from a particular service. Finally the method getOutputStates(Service, InputState) returns output states attainable by a particular Service provided with a particular input State.

The transaction manager is provided with a path and then will claim all needed resources and control invocation of all application services being part of the path for the purpose of actually carrying out the job.

The mobility manager manages the migration of an application service from one host to another.

Other middleware components needed to manage the distributed environment regarding to binding or creating proxies are known in the art and will not further detailed here.

JOB SUBMISSION HANDLER

In the embodiment shown in Fig. 1 users ~~working at a workstation as represented in Fig. 4~~ may define and submit printjobs from one of the connected workstations as represented in Fig. 4. Heretofore a workstation is provided with a print job submission handler 405. The print job submission handler is directly by the user operable as a print job submission application or it may be invoked by a printer driver started within another application program like e.g. a word processor program 406. The print job submission handler will be detailed described now with reference to Fig.6, Fig.7 and Fig.8. Fig.6 gives an embodiment of an operating screen presented to the user to define the job under control of the job submission handler.

~~the objects used by the print job submission handler and Fig. 8 gives a flowdiagram of the steps taken.~~ Fig. 8 is a flowdiagram presenting the steps that are carried out by the job submission handler. Upon activation of a print command or the like by the user the job submission handler is invoked (step 1). After invocation ~~of the job submission handler at step 1~~ in step 2 the operating screen shown in Fig.8 is presented to the user for entry of job parameters. The user defines the job by entering values for the entries in the operating screen

601. The entries: number of copies (602), color printing (603), staple (604), output sheet (605), sorted/collated (606), density (607) and zoom factor (608) all specify the product that will be delivered by the job. The entries price rate (609), security level (610), reliability of service (611), place of delivery (612) and print before (613) are all conditional constraints that as far as they are defined by the user have to be fullfilled. Further constraints are of course possible like e.g. physical distribution with a mailing list. Finally the operating screen also contains a field (613) for entering the file to be printed and a message area (614) for display of messages concerning the job. After acknowledgement of the user that the job definition is by activation of the OK button a Request object R is instantiated. All ~~necessary~~ parameters for the print job are collected inclusive possible default values; and assignment of parameter values to the object instance variables InputState, OutputState and Constraints takes place at intialization of the Request object (step 3).

R = New Request(InputState, OutputState, Constraints).

~~The Request Object as defined above is created in step 3, collected parameters are passed:~~

~~Request R = New Request(InputState, OutputState, Constraints).~~

Next in step 4 the Registry is asked for access to a path composition ~~pathfinder~~ service:

Registry.getService (type = PathComposer).

The Registry returns ~~an ID of a~~ handle for a path compostion service, and ~~. The ID found is passed to a Locator Service, not further explained, and~~ under control of the registry a proxy of the pathfinder at the local station is created.

Subsequently in step 5 the path composition module is asked to retun paths for the defined request:

PathComposer.getPath(R)

The path composition module returns paths that will deliver the product as specified by the job and that fulfils the requested constraints. If more than one path has been found parameters of alternatives disparate of the original request will be shown in the message area, and the user will select one of the alternatives.

If no path has been found (N, step 6), the path composition system returns a message that the job cannot be executed under the requested constraints, and indicates next best paths with relaxed constraints (step 7). The user may select one if one is acceptable for him. If there are no next best paths or the next best paths are not acceptable for the user (step 7, N), the user is asked (step 8) to change parameters belonging to the product specification. If the user responds in the negative (N) the method exits in step 11. If the user accepts this possibility (Y) the method continues with step 9, where all parameters are collected and again in step 3 a new request object is instantiated.

If in step 6 or step 7 a path has been selected and acknowledged (Y) by the user in step 10 the transaction manager is invoked and provided with the path returned. The transaction manager claims application services needed, and invokes subsequently the services needed to fulfil the request. Return messages from the invoked services are received by the Transaction Manager. The transaction manager forwards these messages to the client application, i.e. the job submitter handler. The job submitter handler displays these messages in the message area of the operating screen. When the job has been finished and delivered this is also displayed in the message area of the operating screen. Upon acknowledgement of this message by the user in step 11 the job submission handler logs the data of the job in a logfile and finally the method exits.

PATH COMPOSITION MODULE

(notities gedetailleerde beschrijving: d), (notitie algoritme: e)

The getpath method of the path compostion modile will now be described in detail with reference to the flowdiagram presented in Fig. 9 and the object definitions depicted in Fig. 7.

Task of the getPath method is to find a path of application services that can perform a certain task within given constraints.

To this end the method carries out the following steps:

At invocation in step 1 of the getPath method the Request R is passed. In the Request the input state of the data object to be transformed, the desired target state after the complete transformation and constraints to which the complete transformation has to comply are specified.

In step 2 both a queue object Q and a graph object G are instantiated and initialised. The object Q is instantiated from the class List that supports the methods addElement(), hasMoreElements() and getNextElement(). The method addElement() adds an element to the list. The method hasMoreElements() returns true if the list contains any eements and returns false otherwise. The method getNextElement() returns the next element in the list and removes it from the list.. At initialization the input state of the request will be placed as an element in the queue object Q.

Objects instantiated from the class Graph are used to store and manipulate a directed graph. In our case a node in the graph represents a state of a document and a directed edge starting from a first node going to a second node represents an application service that transforms the state of the first node into the state of the second node. With respect to the request the graph object administers during the execution of the getPath method all states that at a certain moment are attainable and the application services needed for that. For this purpose the class Graph supports the methods setRequest(), addNode(), contains(), connect(), pathFound(), getPaths() and getConstraintRecord(Path). The method setRequest ads a node for the input state and the target state and stores the Constraints expression. The method addNode() adds a node to the Graph. The method connect(Node1, Node2, Service) adds an edge from Node1 to Node2 with reference to the application service that is able to carry out this state transformation. The method contains(Node) checks if a node is contained in the graph. The method pathFound() returns true in case a path has been found. The method getPaths() returns a list of Paths that go from start node to target node. Finally the method getConstraintRecord(Path) returrns a record of values regarding the constraints of concern for the specfied path.

The above results for step 2 in:

```
Q = new Queue()
Q.addelement (R.input)
new G = Graph()
G.setRequest(R)
```

In step 3 it is checked if there are any elements in the Queue. If not the method exits and step 4 is reached. If elements are available (Y) a next input state is fetched from the queue and a list of application services (LAS) that are able to perform a transformation on the input state is requested to the Registry in step 5:

```
currentIn = Q.getNextElement ( )
LAS = Registry.getService(INPUT_STATE == currentIn).
```

Fulfilment of the search criterion for INPUT_STATE is checked with regard to the 'conversion-supported' attribute of the services as listed in Fig.3.

In step 6 it is checked if the List of application services LAS contains any elements. If not (N) the method continues with step 3 to continue with a next input state. If so (Y) in step 7 a next

application service is taken from the list and a List of output states delivered by the Application Service at hand (LOS) is formed:

currentService = LAS.getNextElement ( )

LOS = Registry.getOutputStates(currentService, currentIn)

In step 8 it is checked if a next element is available in LOS. If not (N) the method continues with step 6 to continue with a next application service. If so (Y) in step 9 an output state is taken from the list:

currentOut = LOS.getNextElement

and it is checked if this state has already been reached earlier:

G.contains(currentOut)

If not (N) the method continues with step 10 where the state is added to the Queue and added as a node to the Graph:

Q.addElement(currentOut)

G.addNode(currentOut)

Next step 11 is carried out. In case the outcome of step 9 is positive (Y) the method continues directly with step 11. In step 11 the current application service is added as edge to the Graph

G.connect(currentIn, currentOut, currentApplicationService)

In step 12 it is checked if a path from input to target exists

G.pathFound()

If this is not the case (N) the method continues with step 8.

<<< OPM: wordt er een pad gereturned dat niet noodzakelijk aan de constraints voldoet?

zo werkt het flow diagram wel; en dat mag ook want je zit hier in de geetPath method van de parthfinder. Dus het object graaf retourneert paden die wel van in-> out gaan, maar nog zonder contstraints te checken. Het checken van de constraints gebeurt hier in de getPath method.

>>>

If a path has been found (Y) the method continues with step 13. In step 13 paths found are requested from the Graph and for each path a constraintRecord is fetched. The constraintRecords are compared with the requested constraints and if fulfilment exists that path is selected and in step 14 selected paths are returned and the method exits in step 15.

vvvvvvvvvvvvvvv

ConstraintRecord nog definieren:

Constraints in Request is een expression die true of false oplevert na evaluatie.

ConstraintRecord is een lijst met voor elke constraint een waarde. Dit record wordt door graph object gemaakt door voor elke constraint een optelling langs een pad te maken. Voor elk constraint parameter komt er dus een waarde uit.

Invullen van deze waarden in de constraint request expression levert true of false op.

Vervolgens kunnen de constraint records gerankt worden en de paths met deze ranking uitgegeven worden.

^^^^^^^^^^^^^^^^6

If none of the paths found fulfil the constraints the method continues with step 16. In step 16 a counter N is incremented by one (giving the number of iterations) and the number of found paths is added to a counter M. Further the paths are placed in a list of paths LD.

LD. addelement(path)

Exceeds the counter M a threshold MM, or exceeds the counter N a threshold NN, or are OS, as well as LAS as well as Q empty (Y) then the method will proceed with step 17; otherwise (N) the method continues with step 8 and the Graph G will be further extended. A search for other paths then takes place.

If the method proceeds with step 17, meaning that a maximum number of paths have been found or that the maximum number of iterations have been exceeded, constraintRecords of the paths found are fetched and a List of Paths, orderered with respect to an attribute of the constraintRecord are returned to the requester. Hereafter in step 15 the method exits.

GRAPH CLASS

An embodiment of a Graph class as used in the system according to the invention will be described with reference to Figs 5 and 9.

A graph object instantiated from the class Graph represents a directed graph that is created and intialized in step 2 of Fig. 9 and extended during the repeated executions of steps 10 and 11 in Fig. 9. In the graph a node represents a state and a directed edge starting from a first node going to a second node represents an application service that transforms the state of the first node into the state of the second node. With respect to the request the graph administers all states that at a certain moment are attainable and the application services needed for that. Initially the graph comprises a start node and a target node without an edge.

Data elements of a graph object that are accesible for the outside are: States, Request, Nodes, Edges, Services and Path , and methods: setRequest(), addNode(), contains(), pathFound(), connect() and getPath().

Internal elements of the graph are:

a vector v which elements represent the nodes of the graph

(Moet dit een vector zijn of kan het ook een set zijn; voor de matrix heb je wel een (arbitrair) geordende set van nodes nodig),

a matrix of connections C with elements c_{ij},

a reachableSet containing states that are reachable from the initial state, and

a transformableSet containing states that can be transformed to the target state.

An element v_i of the vector v corresponds with node n_i of the graph. Each node represents a state. A state is represented by a vector s. Element s_i of a state represents a parameter i. An information object has a state that is determined by the values of a number of parameters. In the embodiment here presented the state comprises the parameters: document format, medium, finishing, and location.

Edges, representing a transformation from one state to another, are stored in matrix C. Each element (i,j) provides transformations to come from the state of node i to the state of node j.

The formation of a graph object will be further explained now along the flow of method invocations as presented in the getpath ( ) method with reference to Fig. 9.

In step 2 the Graph object is created by invocation of the statement:

new G = Graph()

and subsequently initialised by invocation of the method G.setRequest(R).

Parameters passed are: the initial state I, the target state T and constraints B. The vector v gets two elements: I and T. The matrix C is initialised with only null values. The reachableSet is initialised with the initial state I and the transformableSet is initialised with the target state T.

In step 9 the method G.contains(node) checks if 'node' is available as an element in the vector v. If so True is returned, if not False is returned.

In step 10 the method G.addNode(node) adds the node to the vector and the matrix is updated by adding an additional row and an additional column.

In step 11 the method G.connect(currentIn, currentOut, currentApplicationService) updates the matrix by looking up the index for currentIn and for currentOut from the vector and subsequently adds the currentApplicationService to the position with this index. Next the reachableSet and the transformableSet are updated: currentOut is added to the reachableSet and it is checked if the node currentOut is present in the transformableSet. If so all parent nodes of currentOut are placed in the transformableSet. If this is the case the Flag Pathfound is set and the method carries out a graph traversal algorithm, known in the art, to find available paths. The paths are added to the List of Paths as far as they are not already contained in the list and the method exits.

In step 12 The method G.Pathfound() returns the value of the flag Pathfound.

In step 14 and 17 the method G.Getpaths() returns the List of Paths.

EXAMPLE

Given are the application services as defined in FIG.4 and the image forming system as shown in FIG.1. In the embodiment in this example a state is defined as a collection of explicitly typed parameters.

At workstation 10 a user specifies a print command resulting in a screen as shown in FIG.6. The format of the printfile generated by the application on workstation 10 is emf (enhanced metafile).

According to the flowdiagram of **FIG. 7**, within the job submission handler in step 3 a request object is created where

source state s_{source} = (format: emf, size: 1 Mb)

target state s_{target} = (format: A4, stapling:Y, number=100, sorted/collated = sorted)

constraints c = (price rate =< 15 $)

Request R = New Request(s_{source} , s_{target}, c)

So the input state is defined by format = emf; the output state is defined by format A4, which implies printing, stapling and number of copies 100; and the constraints are defined by a price rate of at most 10 $.

Next in step 4 a pathfinder service is looked up. A pathfinder service is available at Server 7 and subsequently a proxy for the pathfinder service is created at the workstation 10. Next in step 5 the pathfinder is invoked with the request.

Pathfinder.getpath(R)

According to Flowdiagram in **FIG. 9** upon reception of the request in step 2 a queue Q and a graph G are created and initialised:

This result for the queue in:

$Q = \{ s_0 \}$, where $s_0 = s_{source}$ = (format: emf)

and for the graph G:

Internally for the Graph object this means with reference to FIGS. 11-12:

$\bar{v} = (s_0)$

$C = \{0\}$

$targetState = s_{target}$

In step 5 of the Pathfinder method first an element from the queue is taken

currentIn = s_0

Subsequently application services are looked up that are able to perform a transformation on the input state. A ListofApplication services LAS is in the end returned:

LAS = { as5 }

After a check in step 6 that LAS contains any elements (Y) in step 7 as5 is taken from the list, and a List of output states deliverable by the application service is formed:

currentApplicationService = as5

LOS = { s_1, s_2 }

where s_1 = (format: PCL) and s_2 = (format: HPGL)

After checking in step 8 if LOS contains any elements (Y) in step 9 an element is taken from the list and it is checked if this element is already a node in the Graph:

currentOut = s_1

G.contains (s_1)

This returns False (N) so in step 10 s_1 is stored as a node in the Graph and it is added to the queue Q:

Q.addElement(s_1)

G.addNode(s_1)

Thereafter in step 11 the transformation from state currentIn to state currentOut by application service currentApplicationService is stored as a directed edge in the graph:

G.connect(s_0, s_1, as5)

Internally for the Graph object G the execution of steps 10 and 11 means that at first the reachableSet and the transformableSet are updated. Subsequently the node is added to the vector and the dimension of matrix C is adapted accordingly. Next the method G.connect() updates the matrix by looking up the index for currentIn and for currentOut from the vector and subsequently adds the currentApplicationService to the position with this index. Herafter the method checks if the node currentOut complies with the Target node. This is not the case.

With respect to the internal members of G this results in:

$\vec{v}$ = (s_0, s_1)

$$C = \begin{matrix} 0 & as5 \\ 0 & 0 \end{matrix}$$

Pathfound: False

In step 12 the flag pathFound is checked. This yields 'false' (N) so that the method continues with step 8.

The element s3 is available in LOS (Y) so in step 9 this element is taken from the list and the same sequence of steps 9 – 10 – 11 is carried out as explained before.

This yields for Q = { s_1, s_2 }

and for the internal members of G this results in:

$\vec{v}$ = (s_0, s_1, s_2)

$$C = \begin{matrix} 0 & as5 & as5 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Pathfound: False

In step 12 the flag Pathfound is checked. This yields again 'false' (N), so the method continues with step 8. In step 8 it is checked if LOS contains any output states. That is not the case (N) so the method continues with step 6. In step 6 it is checked if the list LAS contains any elements. This is not the case (N). The method continues with step 3. In step 3 it is checked if the queue Q contains any elements. This is indeed the case (Y). In step 5 the next element from the queue is taken and a listOfApplicationServices is returned for this state:

currentIn = s_1

LAS = {as3, as4}

After a check for the presence of an element in the List in step 5 (Y) in step 7 as3 is taken from the list and a list of output states delevirable by this application service is formed:

currentApplicationService = as3

LOS = {s_3} where s_3 = (format: Postscript)

Subsequently steps 8 – 9 –10 – 11 are carried out resulting in:

Q = { s_2, s_3 }

$\vec{v}$ = (s_0, s_1, s_2 ,s_3)

$$C = \begin{matrix} 0 & as5 & as5 & 0 \\ 0 & 0 & 0 & as3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

Pathfound = false

In step 12 the flag Pathfound is checked. This yields again 'false' (N), so the method continues with step 8. In step 8 it is checked if LOS contains any output states. That is not the case (N) so the method continues with step 6. In step 6 it is checked if the list LAS contians any elements. This is indeed the case (Y). The method continues with step 7 with as4 as currentApplicationService. This ApplicationService yields as output states:

LOS = {s_4} where s_4 = (format: PDF)

After processing in steps 8 – 12 this results in:

Q = { s_2, s_3, s_4 }

$\vec{v}$ = (s_0, s_1, s_2 ,s_3, s_4)

$$C = \begin{matrix} 0 & as5 & as5 & 0 & 0 \\ 0 & 0 & 0 & as3 & as4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Pathfound = false

The method continues until eventually s_3 is next from the gueue Q in step 3.Carrying out of steps 5-6-7 yields:

LAS = { as1}

currentApplicationService = as1

LOS = { s_5 } where s_5 = (sheet = A4, color = Yes, PPM = 70, Staple = Yes)

currentOut = s5

Finally the following situation is reached:

Q = { s_4, s_5, }

$\vec{v}$ = (s_0, s_1, s_2 ,s_3, s_4, s_5)

$$C = \begin{matrix} 0 & as5 & as5 & 0 & 0 & 0 \\ 0 & 0 & 0 & as3 & as4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & as1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Pathfound = True since s_{target} is a subset of s_5 : $s_{target} \subseteq s_5$

The resulting graph is depicted in Fig. 10.

The method continues with step 13.

In step 13 the path found is requested from the Graph:

G,getpath yields:

Path = (as5, as3, as1)

Costs are calculated:

as5 costs: 1 $

as3 costs: 1$

as1 costs: 10 $

This gives as total: 12 $

So constraints will be fullfilled.

In step 14 the pathfinder will return the path to the job submission handler.

The user acknowledges the path and in the next step the transaction manager is invoked and provided with the path found. The transaction manager claims the application services. If each application service has committed itself under control of the transaction manager the application services are executed in sequence.

In case the constraints are not fulfilled the pathfinder service continues with step 16 where it is checked if a counter exceeds a threshold. This is not the case (N). The method continues with step 18. In step 18 the The method continues with step 8. No further output states are available in LOS (N). So the method checks for further application services in step 6. No further application services are available in LAS (N). Subsequently the method checks for a next state in the queue Q: next state is s4. The processing of s4 as currentIn yields s3 as currentOut. This node is already available in the transformableSet so a path is found.

The new path found now is: as5 – as4 – as2 – as1.

Compliance with constraints is again checked.

In the case there is no compliance again in step 14 in step and LOS, LAS and the queue Q are all empty.

Having now fully described the invention, it will be apparent to one of ordinary skill in the art that many changes and modifications can be made thereto without departing from the scope of the appended claims.

It is noted that the invention is not restricted to text documents but the invention is equally applicable to all kinds of documents e.g. documentws that contain only or in combination video information and audio information.

(Example 2)

EERSTE CLAIM SET

1 tertis Distributed document handling system for carrying out jobs,
where jobs are carried out by services distributed over a network and where a job leads to a product as a result from a production process,
comprising:
a pool of services the services being distributed over a number of interconnected processing devices;
means for entering a job specification comprising product specifications specifying the product to be delivered by the job; and
means for determining a path of services, the services being selected from the pool of services, suitable to carry out the job in accordance with the product specifcations, c.i.t.
the job specification also comprises specifications specifying circumstantial constraints without effect on the product; and in that the means for determining the path of services take into acount circumstantial constraints for that job.

Afbakening van Sharp

Sharp heeft geen constraints die betrekking hebben op circumstances; heeft alleen constraints om ervoor te zorgen dat beeldkwaliteit over hele document goed is.

is dus beperkt in de mogelijkheden; je wilt een pad kunnen maken waarbij ook rekening gehouden wordt met prijs, betrouwbaarheid, security, afstand, zaken die onafhankelijk zijn van het uiteindelijke product.

2. Distributed document handling system according to the previous claim c.i.t.

a circumstantial constraint defines a limit in an ordered range; and in that

the system also comprises means for ranking paths suitable to carry out the job in accordance with the ordered range of the circumstantial constraint and user interface means for selection by the user of a desired job specification from a ranked list of job specifications based on the ranked paths.

3. Distributed document handling system according to the previous claim c.i.t.

the system comprises user interface means for selection by the user of the circumstantial constraint to be used in the ranking of the paths.

4. Distributed document handling system according to claim 2 c.i.t.

the circumstantial constraint is a total price of the job to be carried out and in that the system comnprises means for calculating the total price from price attributes of services comprised in a determined path.

[OPM: in 1 nog niets gesteld over de hoeveelheid oplossingen die de middelen opleveren; kan er dus ook meer dan een zijn;

Onderconclusies:

minimaliseren op bp. constraint; ranken en aanbieden

te minimaliseren constraint wordt ingegeven door de gebruiker
geld: calculation of total

TWEEDE SET

1 [tertS] Distributed document handling system for carrying out jobs, where jobs are carried out by services distributed over a network and where a job leads to a product as a result from a production process, comprising:
a pool of services the services being distributed over a number of interconnected processing devices;
means for entering a job specification comprising product specifications specifying the product to be delivered by the job; and
means for determining a schedule of services, the services being selected from the pool of services, suitable to carry out the job in accordance with the product specifcations,
c.i.t.
the system also comprises means for, in case no suitable schedule of services can be determined, migrating a service from one device to another in order to enable the determination of a schedule of services in comliance with the job specification.

2. Distributed document handling system according to the previous claim comprising means for registering a service in a register; means for transmitting a service from one node to another; and means for updating the register accordingly.

TOELICHTING:

Geen oplossing dan migreren van services

Is te gebruiken voor elke systeem waarbij een digraaf gemaakt wordt; staat los van al dan niet circumstantial constraints

Sharp: als nadeel dat als bewerkingen buiten, buiten bedrijf, buiten firewall, plaats vinden er altijd een security lek is. Scramblen (zie Fig.15) is alleen in enkele gevallen mogelijk, niet bv. bij vertalen, bij image processing etc.

DERDE SET

1 tertis Distributed document handling system for carrying out jobs,
where jobs are carried out by services distributed over a network and where a job leads to a product as a result from a production process,
comprising:
a pool of services the services being distributed over a number of interconnected processing devices;
means for entering a job specification comprising product specifications specifying the product to be delivered by the job; and
means for determining a schedule of services, the services being selected from the pool of services, suitable to carry out the job in accordance with the product specifcations,
c.i.t.
the system also comprises means for, in case no suitable schedule of services can be determined, relaxing the job specification in order to enable the determination of a schedule of services in compliance with the relaxed job specification.

2. Distributed document handling system for carrying out jobs according to the previous claim, c.i.t.
the job specification comprises also circumstantial constraints and in that the means for relaxing the job specification relax the conditional constraints.

3. Distributed document handling system for carrying out jobs according to the previous claim, c.i.t.
the system also comporises means for presenting a relaxed job specification to the user for acknowledgement.

Onafhankelijk: geen oplossing dan constraints relaxen; ranken en aanbieden

Onafhankelijk, immers ook te gebruiken als er alleen product specifications zijn

onderconclusie: eerst proberen om circumstantial constraints te relaxen, daarna andere constraints relaxen

Geen oplossing dan eerst zoeken naar pad met relaxed circumstantial constraints; ranken en aanbieden;

bruikbaarheid van het systeem wordt groter; voor onderconclusies (circumstantial constraints: hierover valt te onderhandelen

(18-8-00 discussie claims)

[redacted]

nieuw: je hebt een sequentie van services
om met de constraints om te gaan moet je nu constraints per service b
elke service levert een bijdrage tot de constraints
uit elke bijdrage moet je totaal berekenen en vergelijken met waarde voor de job
constraitns voor hele pad afleiden uit de afzonderlijke bijdragen van de services
means for calculating total based on distinct contributions of services
means for determining total constraint based on distinct contributions of services

waar zit verschil i.v.m. conv. reproductie-apparaat: afzonderlijke services; elke service heeft eigen bijdrage tot constraints; constraint van iedere service moet in beschouwing worden genomen
[andere printer kiezen: dan wijzig je een production constraint

[redacted]

[andere printer kiezen: dan wijzig je een production constraint

[redacted]

1 tertis Distributed document handling system for carrying out jobs,
where jobs are carried out by services distributed over a network and where a job leads to a product as a result from a production process,
comprising:
a pool of services the services being distributed over a number of interconnected processing stations;
means for entering a job specification comprising product specifications specifying the product to be delivered by the job; and
means for determining a schedule of services, the services being selected from the pool of services, suitable to carry out the job in accordance with the product specifcations,
c.i.t.
the job specification also comprises specifications specifying circumstantial constraints without effect on the product; and in that the means for determining the schedule of services take into acount circumstantial constraints for that job.

The term **path of services** is used to denote the set of services that are involved in realizing the requested job and which are not necessarily a set of services that are executed in sequence but may also encompass services that have to be executed in parallel.

ABSTRACT

Networked reproduction system, comprising a number of nodes with connected scanners, printers and servers.

A reproduction job to be carried out is composed of a number of subtasks. For the execution of these subtasks services distributed over the network are available. A service manager selects appropriate services and links them to form paths that are able to fullfill the reproduction job. a path, optimal with respect to constraints, is selected.

Gebruikte begrippen:

Service manager	pathfinder
Application service	
Core Service	component

beschrijving beperkt tot de componenten die strikt nodig zijn voor pathfinder services:

service request handler
pathfinder
registry: registreert/deregistreert objecten
hier ook objecten op te vragen aan hand van requirements
lokaliseert ook objecten aan hand van GUI

transaction manager:
voor het uitvoeren van een pad
core:
creeert een VM over alle peripherals heen, dus alle appropriate services zijn remote accessible. Dus niet iets expliciet nodig voor binding of proxy o.i.d.

mobility manager: verplaatsen services
nog even laten staan voor zover migration nog onderwerp kan worden van de aanvrage

Niet strikt nodig:
distribution manager:
services lokaal uitvoerbaar maken via proxy
elke service is remote accessible
user agent/service agent: taak vooral overgenomen door service request handler
mobility manager: verplaatsen services
trading service
locator

Bookmarks

a

beschrijving beperkt tot de componenten die strikt nodig zijn voor pathfinder services:

service request handler
pathfinder
registry: registreert/deregistreert objecten
hier ook objecten op te vragen aan hand van requirements
lokaliseert ook objecten aan hand van GUI

transaction manager:
voor het uitvoeren van een pad
core:
creeert een VM over alle peripherals heen, dus alle appropriate services zijn remote accessible. Dus niet iets expliciet nodig voor binding of proxy o.i.d.

mobility manager: verplaatsen services
nog even laten staan voor zover migration nog onderwerp kan worden van de aanvrage

Niet strikt nodig:
distribution manager:
services lokaal uitvoerbaar maken via proxy
elke service is remote accessible
user agent/service agent: taak vooral overgenomen door service request handler
mobility manager: verplaatsen services
trading service
locator

b

DETAILED DESCRIPTION OF SHARP

[3/4] Image information, corresponding to a plurality of pages is inputted to one of the image forming apparatuses. When receiving instructions to carry out a predetermined image processing with respect to the image information, some of the image forming apparatuses carry out the image processing in a distributed manner with respect to the image information for every predetermined pages of the image information in parallel and within processing capacity levels of the some image forming apparatuses, the processing capacity levels falling within a permissible range.

[3/41] Image processing sections in other apparatuses carry out the image processing that has been specified in accordance with the control information, where after the image information that has been processed is returned to the first image information apparatus. [4/5] There it is visualised by the image output section of the first image information apparatus.

[4/14] In the system according to Sharp it is not necessary that image processing functions that an operator intends to are provided in a first image processing apparatus, such an image processing can be carried out by other image forming apparatuses. And accordingly it is not necessary for each image forming apparatus to have separately predetermined image processing functions therein.

[5/18] A predetermined image processing is carried out in parallel by the first and second image forming apparatuses, such an image processing can be quicly carried out.

[9/18] Each of the second image forming apparatus further transmits to another one of the second image forming apparatuses the image information inputted through the communication apparatus when each of the second image forming apparatuses can not carry out the image processing, [9/24] e.g. when the second image forming apparatus is in inoperable condition such as in trouble condition.

[10/53] The present image forming system is arranged such that a plurality of digital copying machines are operatively connected through a communication apparatus.

[11/54] the image processing section is provided with an image data input section, an image data processing section, an image data output section, a memory and a print control unit that functions as a control section.

[[Fig.11]: one of the functions e.g. character recognition, is not available on all machines.

[16/45]: network e.g. by daisy chain, Ethernet.

[17/42]: (available) function information is stored in memories, this is done by exchanging function information between the copying machines. (Opm: dit in tegenstelling ot een centrale registry)

[18/46]: using general purpose networks, hackers, scrambling.

[19/29] as metnioned above, according to the present image forming system, when the image processing functionthat the digital copying machine91 does not have is selected, such a selected image processing functioncan be carreid out in a distribured manner by the other dirigal copying machines. Accordingly, it is not necessary for all ghe digiral cipying machines in hte image forming systenm to have excellent image processing functions, respectively.

[19/37] It is also possible to select the machines to which the already processed image data is returned, depending on various conditions so that the selected digital copying machines or their printers output an image.

[20/28] request to carry out sharpness processing

[21/31] an operation for judging of edges of mage (herinafter referred to an an image edge hudging operation) is provided as an example.

Notes tot 25/45

Nadelen/Beperkingen stand techniek/Voordelen uitvinding.

c

DETAILED EXPLANATION OF JAVA

n--------addition 29-9-99--------

A description is given in The Java Language Environment, A White Paper, by J.Gosling and H.McGilton, both of Sun Microsystems and in 'The Java Platform", by Douglas Kramer of Sun.

Java consists of an interpreted object-oriented programming language and a supporting runtime environment. In the Java language a number of objects can be defined. Each object has state and behavior. An objects behavior is defined by its methods.

[35] A class is a software construct that defines the data (state) and methods (behavior) of the specific concrete objects that are subsequently constructed from that class. In Java terminology a class is built out of members, which are either fields (for data) and methods. A class is a template for an object. You obtain a concrete object by instantiating a previously defined class.

[38] One object can invoke a method of another object for manipulating the data of that other object. In this way you can build a whole system of objects that interact with each other.

[44] In the Java language the notion of 'Interface' is introduced. An interface in the Java language is simply a specification of methods that on object declares it implements.

[46] packages are collections of calsses and interfaces that are related to each other in some useful way.

[51] The Java compiler generates bytecodes: a high-level machine-independent code for a hypothetical machine that is implemented by the Java interpreter and run-time system.

[52] So compiled Java language programs are portable to any system on which the Java interpreter and run-time system have been implemented. The Java environment itself is readily portable to new architectures and operating systems.

[56] The Java language virtual machine is a strictly defined virtual machine for which an interpreter must be available for each hardware architecture and operating system on which you wish to run Java language applications.

[57] Classes are linked in as required and can be downloaded forn across networks. Incoming code is verified before being passed to the interpreter for execution.

[64] Java's networking package provides the interfaces to hadnle the various network protocols.

[66] You can write multithreaded applications.

[76] The complete Java system includes several libraries of utility classes and methods of use to developers in creating multi-platform applications.

[java platform]

[24] A Java language development environment includes both the compile-time and runtime environments (Figuur 1a). The Java platform is represented by the runtime environment. In there any calsses from the Java Class Libraries (API) are dynamically loaded as needed by the applet.

[6] The java platform is a new software platform for delivering and running highly interactive, dynamic and secure applets and applications on networked computer systems. While each underlying platform has its own implementation of the Java virtual machine, there is only one virtual machine specification.

[7] Programs written in the java language and then compiled wil run on the java Platfrom. The Java platform has two basic parts: - The java virtual machine – Java application programming interface. ([22] an API is a collection of classes; noot: deze klasses kun je ggebruiken in je programma. Het zijn Java klassen en dus verder onafhankelijk van het onderliggende o.s.

[9] Two different kind of programs: applets (require a browser to run, have a built-in downloading mechanism) and applications (programs that require a browser to run)

[12] The java language is the means for a developer to write sourece code. Applets and applications written in the Java language compile to a form that runs on the Java Platform. When developers write source code in the Java Language , this code can call API's defined in the Java base API, java standard extensions API, or a new API defined in the source code itself. At runtime all three kinds of API's have equal standing.

[14] The java platform: two main parts: the Java Virtual Machine and the Java API.(figuur 1b). The Java API forms a standard interface to applets and applications, regardless of the underlying operating system. The Java API is the essential framework for application development. API specifies a set of essential interfaces in a growing number of key area's that developers will use to build their Java-powered applications.

[15] The classes are the implementation of that API. The Java API framework is open and extensible. specifications for each interface are being developped. The API is organized by groups, or sets. Each of the API sets can be implemented as one or more packages (namespaces). Each package groups together a set of calsses and interfaces that define a set of realted fields, constructors, and methods. The java Virtual machine is a platform that hides the underlying operating system from Java-powered applets and applications. In addition the VM defines a machine-independent format for binary files called the class (.class) file format. This format includes instrucctions for a virtual conmputer in the form of bytecodes.

[16] The Java base API (also known as java core API, or java applet API), defines the basic building blocks for creating fully functional java-powered applets and applications. It includes all the classes in the java-package: java.lang, java.util, java.io etc.

[21] Java server API is an extensible framework that enables and eases the development of a whole spectrum of java-powered internet and intranet servers. The framework also encompasses the Servlet API. Servlets are platform-independent Java powered objects, the server side counterpart of applets. Servlets can reside locally on the server, or be downloaded to the server from the net under security restrictions.

Java programs in source code are compiled into a set of executable bytecodes. These sets of bytecodes can be executed in the java runtime environment. For all kinds of machines and operating systems the java runtime environment looks the same from the viewpoint of the sets of bytecodes.

u--------addition 29-9-99--------

n---------addition2 29-9-99------

Bovenstaande java-omgeving geeft the computational environment for CORE. Thus Core is implemented by writing javaprograms that by itself will run on a variety of heterogeneous linked computers.

The core environment is thus defined in terms of a number of interacting objects (at least in the sense of the java language).

How is this environment organised, what is the architecture

[133]

3 A core system consists of components (called services) that interact with the user and/or each other. When a new service is introduced into the system, it exists as though present in the whole system.

[het volgende zijn meer algemene object-begrippen:

[proxy-principal model: is dat hier nodig?]

[binding van een thread aan een object: is dat relevant?]

[aan hand van uitgewerkte voorbeelden nagaan of bovenstaande begrippen handig zijn in de beschrijving]

d

NOTITIES GEDETAILLEERDE BESCHRIJVING:

Centraal in de beschrijving staan de flowdiagrammen: een aantal stappen die worden uitgevoerd cm het gewenste resultaat te verkrijgen.

Ik vraag me af of het handig is om voor de beschrijving object-georienteerd te werk te gaan. OO zou zo iets betekenen van: er worden een aantal objecten gedefinieerd, de methode werkt met deze objecten. Dit sluit aan bij de beschrijving van de uitvinders.

Concreet: in het flowdiagram worden acties gedefinieerd in termen van de gedefinieerde objecten. Object staat klassiek gesproken dan voor een datastructuur met bijbehorende operaties op die datastructuur.

(Vraag voor mij dan nog is: what makes the objects move?: Wat is de computational environment. Eigenlijk: hoe ziet een OO-programma eruit en merk je nog iets van objecten als het gecompileerd is? etc?). Vergelijken met aanvragen van Sun.

Is een systeem op deze manier beschreven na te werken?

What makes the objects move: dat is toch eiigenlijk een programma, het programma (en in actie dus het proces of de thread) bepaalt wanneer een method van een object aangeroepen wordt, door welk ander object dat aangeroepen wordt, hoe de sequentie van acties op objecten is, onder welke omstanidgheden bepaalde acties worden uitgevoerd, etc. Voor mij is inderdaad de vraag: hoe vind je de objecten terug in de programmaregels)

e

23-11-99

NOTITIES AANPAK

[redacted]

[redacted]

EXAMPLE 2

In this embodiment four services are available. Each service is able to carry out one or more file format transformations (which formats, examples)

The following services are registered:

S1(A->B(1$), A->C(5$))

S2(B->D(1$))

S3(B->C)

S4(C->D)

So Service 1 is able to transform file format A into file format B at a price of $1 per file and to transform file format a to format C at a price of $5 per file. Other services are defined in the same manner.

A request is now received by the Service management system:

request(A->D($1.5))

The request asks to carry out a file format transformation from format A to format D at a price less than $1.5.

So in terms of the methods of the object Request:

getInput = A

getOutput = D

getBound = less than $ 1.5

In an intialization step new instance of the Queue Object and of the Graph object are created.

In Queue wordt element toegevoegd: A

Is dat zo, ik zou hier D verwachten

In Graph wordt request gezet: dat betekent dat A en D in de graph worden gezet

In step 3 it is checked if object Queue has elements

If not method stops

If yes then in step 5 a list of services that can handle next format (is next Format uit Q bekend?) Next format is A. Object Trader is asked to return all services that are able to handle format A. Trader returns service S1.

In step 6 it is checked that a service is available.

In step 7 a list of output formats of service S1 is obtained:

In step 8 it is checked if there is at least one

If not: return to step 6

If yes: step 9: get the next output format: B
In step 10 check if format is already in graph
yes: directly to 12
no: in step 11: element is added to Queue and to Graph
step 12: make link from A to B in graph
In step 13 it is checked if within Graph a suitable path exist
yes: in step 14 path is returned
no: return to step 8
in step 8: icheck if there is a next output format for the given input format
yes: a next iteration starts with respect to adding a next output format to the Queue and the Graph starting from the previous input format (i.e.A).
no: the method returns to step 6:
in step 6 it is checked if there are more services availble for the current input format
if yes: output formats and possible paths are added to Queue and Graph
if no: method returns to step 3:
in step 3: it is checked if there are any next formats in the Queue

Vraag me af:
dit flowdiagram geeft maar een pad af?
Checking op boundary conditions gebeurt in object Graph?

Varianten

Eerste indiening gericht op werken met constraints,
waarbij in beschrijving constraints alleen betrekking hebben op vast bedrag voor bepaalde service. Dan later 4) nog mogelijk.

Of: method for the run-time generation of a schedule of services in a dynamic distributed environment

Overige varianten:

1.) Foutafhandeling: als een service zich afmeldt dan alternatief pad kiezen

2.) Service migreren zodanig dat wel aan constraints voldaan wordt
(Bijvoorbeeld service die binnen firewall uitgevoerd moet worden)

3.) Verplaatsen legacy code

4. Verdere uitwerking van het werken met constraints:

niet alleen geld, maar ook relieabiltiy, tijd, confidentiality

Type of Application Service	ID	Attributes
printer	as 1	document-format-supported = Postscript media-supported = iso-a4-white, na-legal-white finishing-supported = punch, staple costs = pay: page: 0,05: $ location = NL 5900 PL 3G
conversion_of_doc umentformat	as 2	conversion-supported = PDF-Postscript costs = pay: Kbyte: 0,01: $ location = NL 5900 PL 3 B
conversion_of_doc umentformat	as 3	conversion-supported = PCL-Postscript costs = pay: File: 0,75: $ location = NL 5900 PL 3 B 25
conversion_of_doc umentformat	as 4	conversion-supported = PCL – PDF, costs = pay: File: 0,75: $ location = NL 5900 AA PostOffice
conversion_of_doc umentformat	as 5	conversion-supported = EMF-PCL, EMF-HPGL costs = pay: File: 0,75: $ location = NL 5900 AA 3 B 25
printer	as 6	document-format-supported = HPGL media-supported = iso-a4-white, na-legal-white costs = pay: page: 0,10: $ location = NL 5900 SJ 3A 35

10-8-00

Issues under discussion:

1. Usage of the term "digraph
2. "Term 'digraph of services is used to denote the set of services that are involved in realizing the requested jogb and which are not necessarily invoked in sequence but which may also be carried out in parallel.

a pool of services

means for composing a digraph of services from the pool suited to carry out a job

attributes associated with a service

constraints associated with the job

Distributed document handling system

comprising:

a number of interconnected nodes connected via a network

services located on the nodes forming together a pool of services

means for entering a job specification

selection means for selecting services from the pool of services for carrying out the job

c.i.t. the system further comprises

means for transmitting a service from one node to another in case the selection means fail to select services order to comply with the job specification

Fig.1

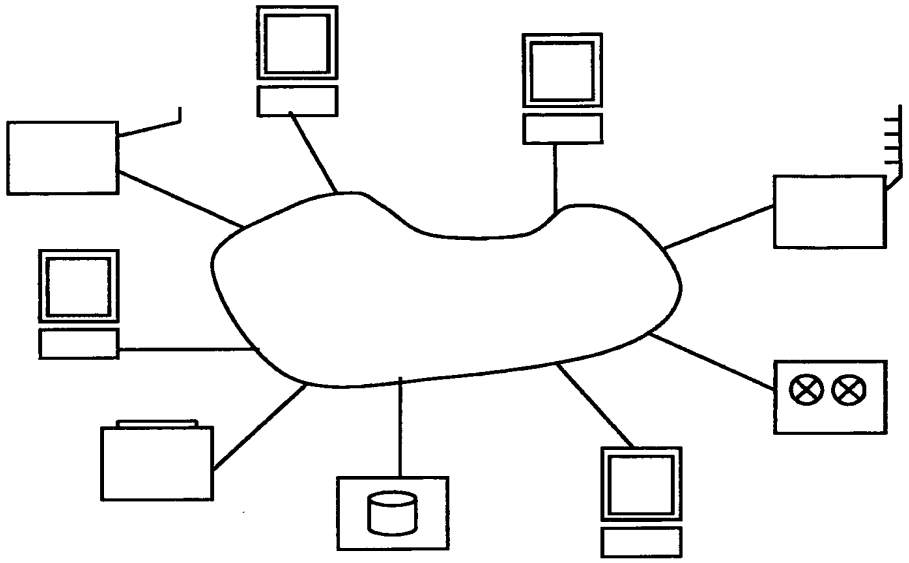

Fig.2

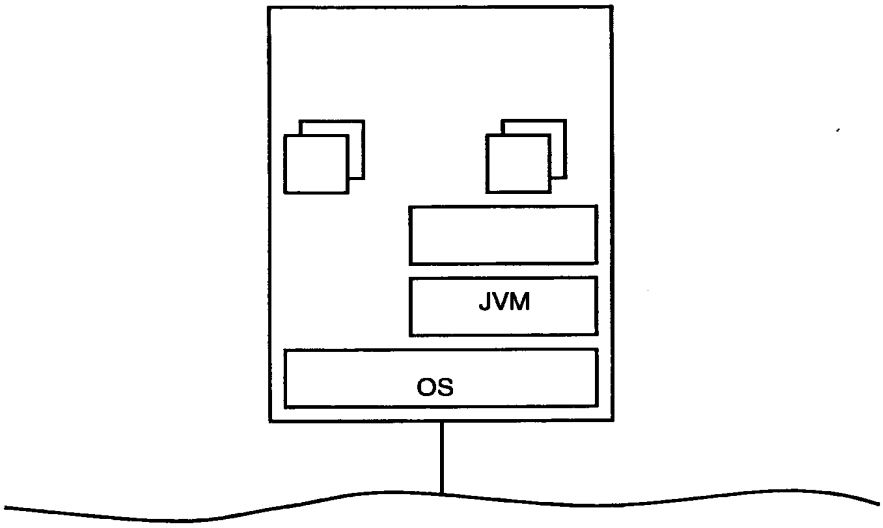

Fig.3

Type of Application Service	ID	Attributes
printer	as 1	document-format-supported = Postscript media-supported = iso-a4-white, na-legal-white finishing-supported = punch, staple costs = pay: page: 0,05: $ location = NL 5900 PL 3G
conversion_of_doc umentformat	as 2	conversion-supported = PDF-Postscript costs = pay: Kbyte: 0,01: $ location = NL 5900 PL 3 B
conversion_of_doc umentformat	as 3	conversion-supported = PCL-Postscript costs = pay: Kbyte: 0,01: $ location = NL 5900 PL 3 B 25
conversion_of_doc umentformat	as 4	conversion-supported = PCL – PDF costs = pay: Kbyte: 0,02: $ location = NL 5900 PL 24
conversion_of_doc umentformat	as 5	conversion-supported = EMF-PCL, EMF-HPGL costs = pay: Kbyte: 0,01: $ location = NL 5900 PL 3 B 25
printer	as 6	document-format-supported = HPGL media-supported = iso-a4-white, na-legal-white costs = pay: page: 0,10: $ location = NL 5900 PL 16 3A 35

Fig.4

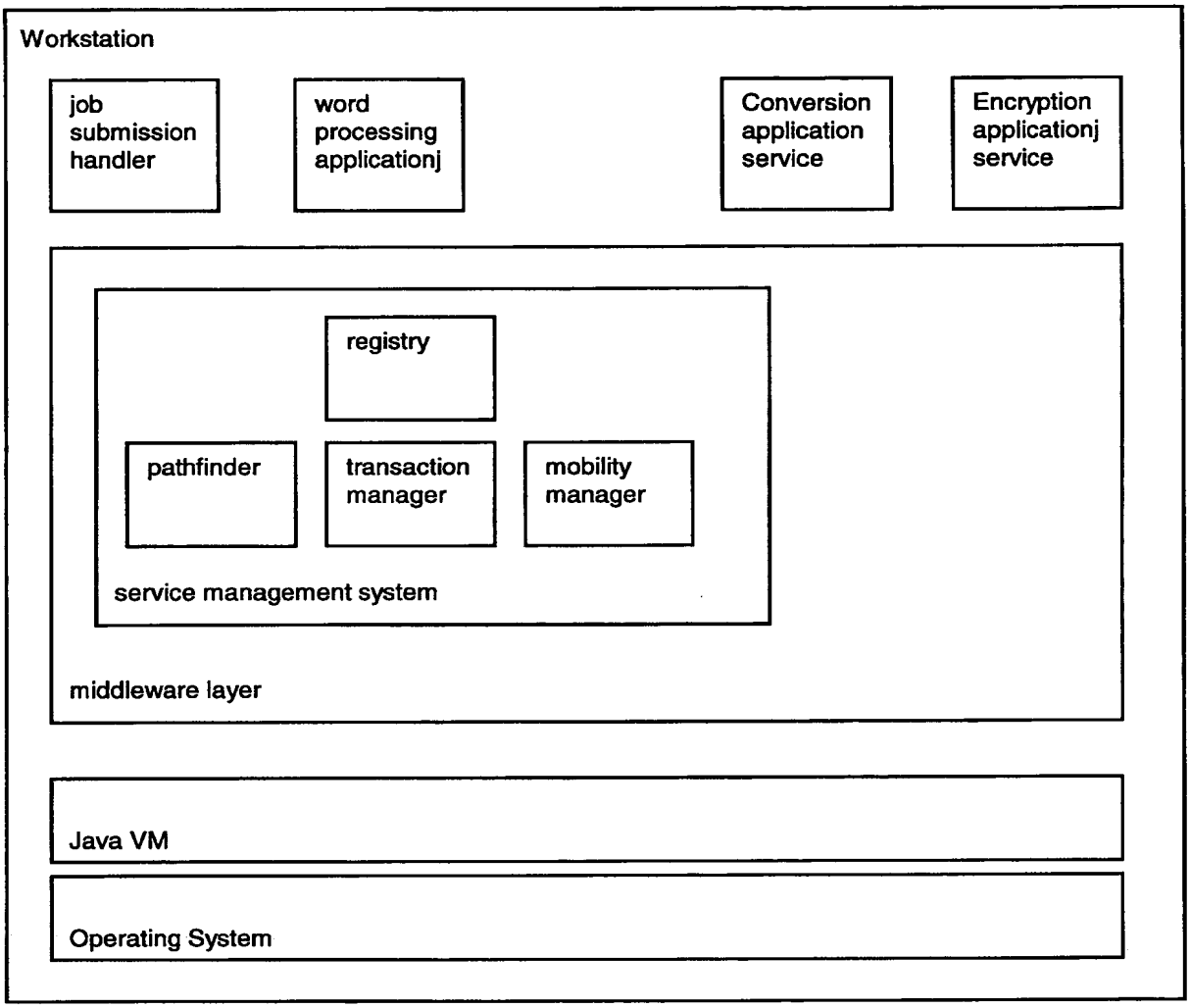
Workstation
job submission handler
word processing applicationj
Conversion application service
Encryption applicationj service
registry
pathfinder
transaction manager
mobility manager
service management system
middleware layer
Java VM
Operating System

Fig. 5

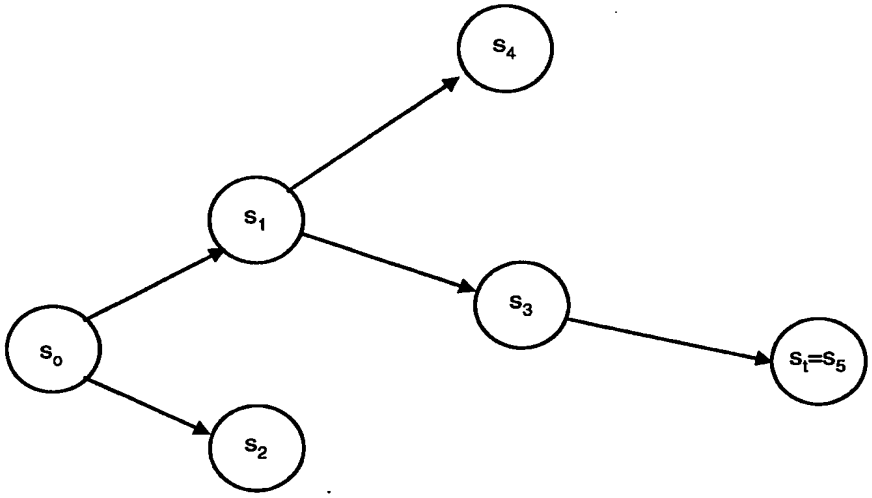

Fig.6
job submission handler

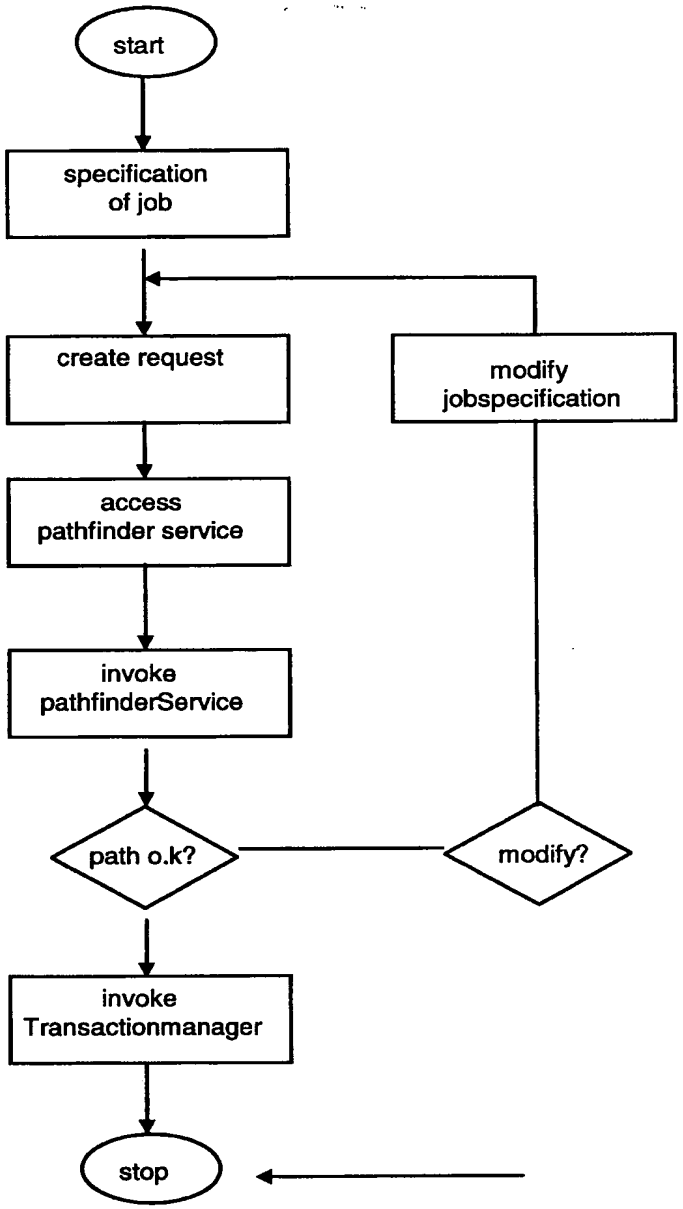

Fig.7

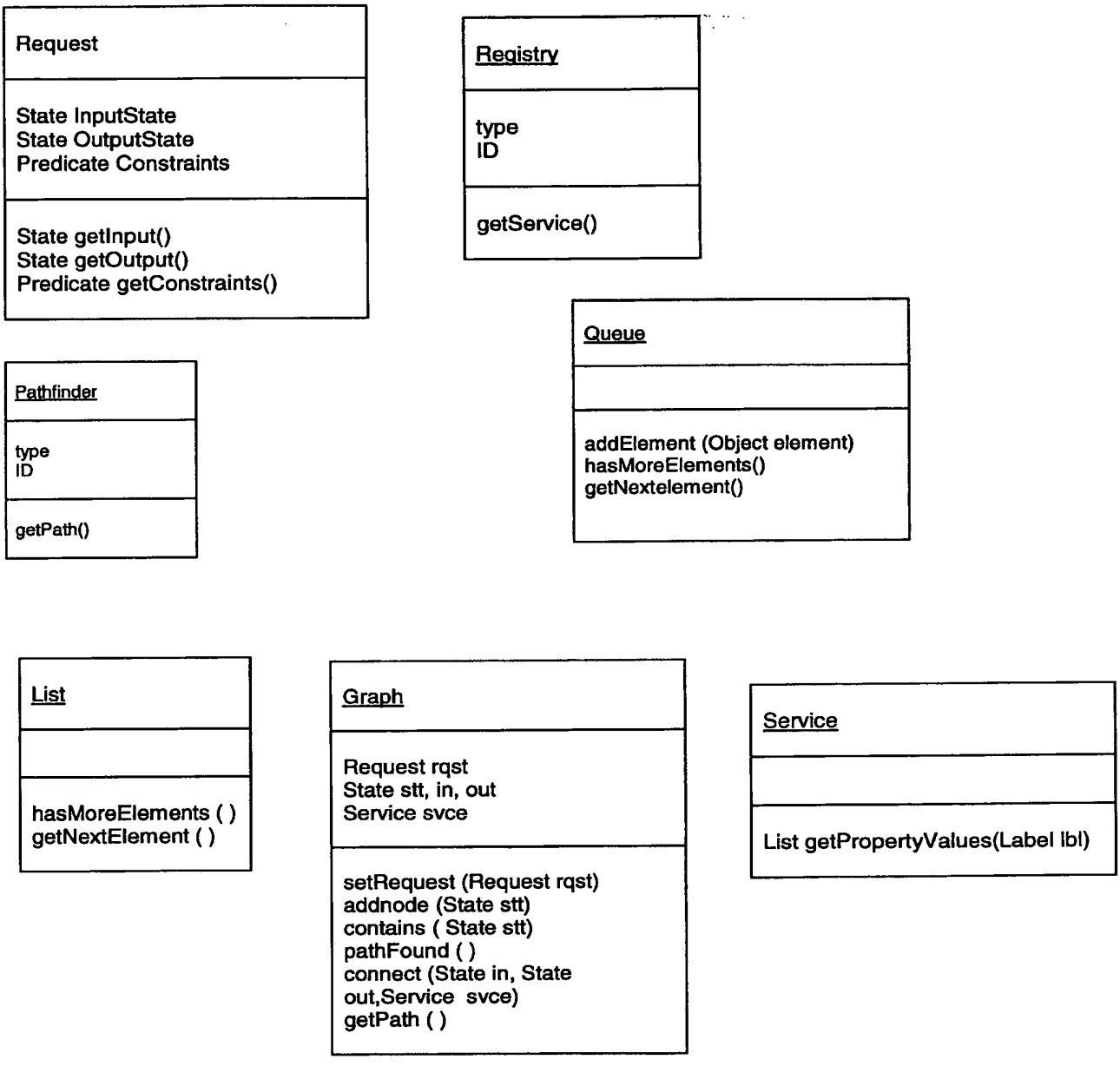
Request
State InputState
State OutputState
Predicate Constraints
State getInput()
State getOutput()
Predicate getConstraints()
Registry
type
ID
getService()
Pathfinder
type
ID
getPath()
Queue
addElement (Object element)
hasMoreElements()
getNextelement()
List
hasMoreElements ( )
getNextElement ( )
Graph
Request rqst
State stt, in, out
Service svce
setRequest (Request rqst)
addnode (State stt)
contains (State stt)
pathFound ( )
connect (State in, State
out,Service svce)
getPath ( )
Service
List getPropertyValues(Label lbl)

Fig.8

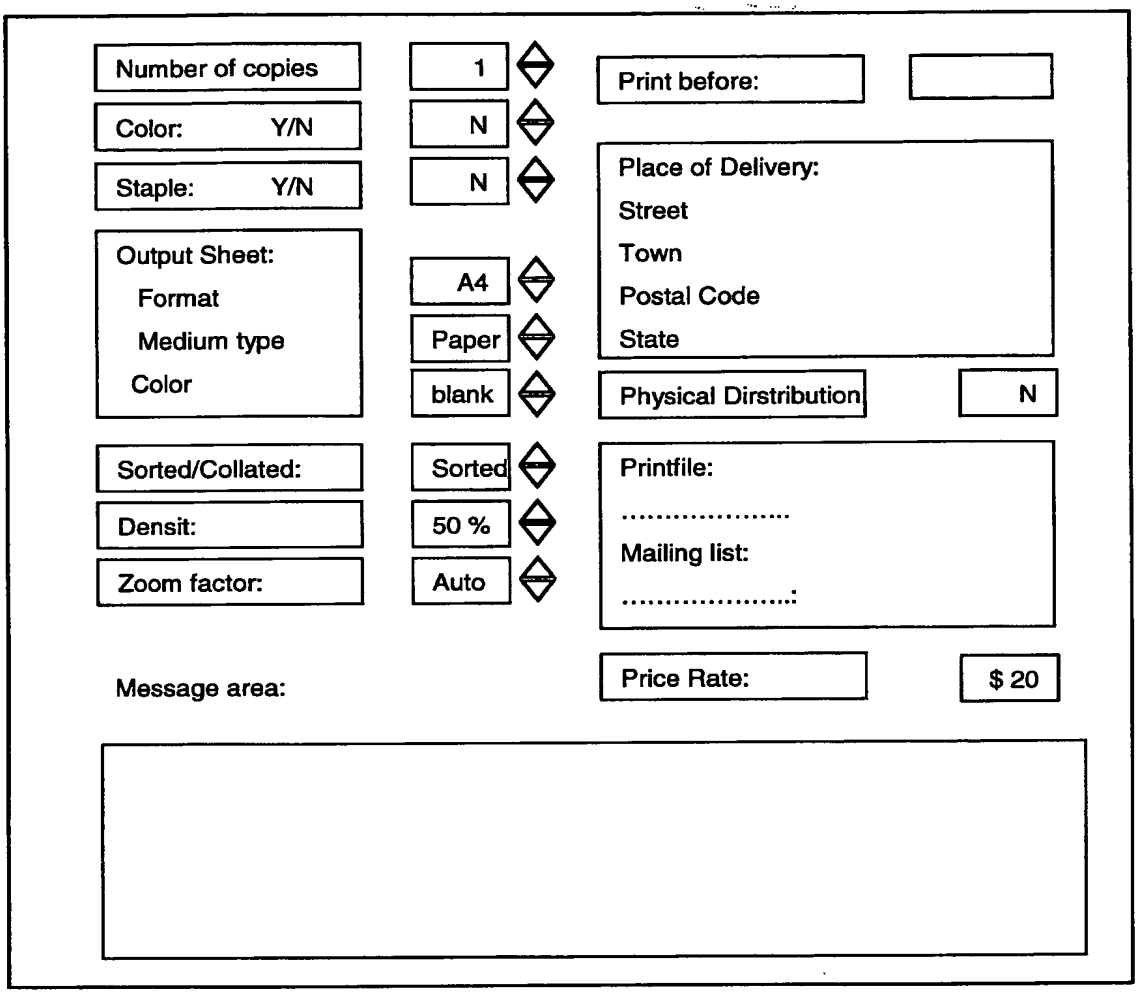
Number of copies
1
Print before:
Color: Y/N
N
Staple: Y/N
N
Place of Delivery:
Street
Town
Postal Code
State
Output Sheet:
Format
Medium type
Color
A4
Paper
blank
Physical Dirstribution
N
Sorted/Collated:
Sorted
Printfile:
.....
Densit:
50 %
Mailing list:
.....
Zoom factor:
Auto
Message area:
Price Rate:
$ 20

Fig.9

Pathfinder.getpath method

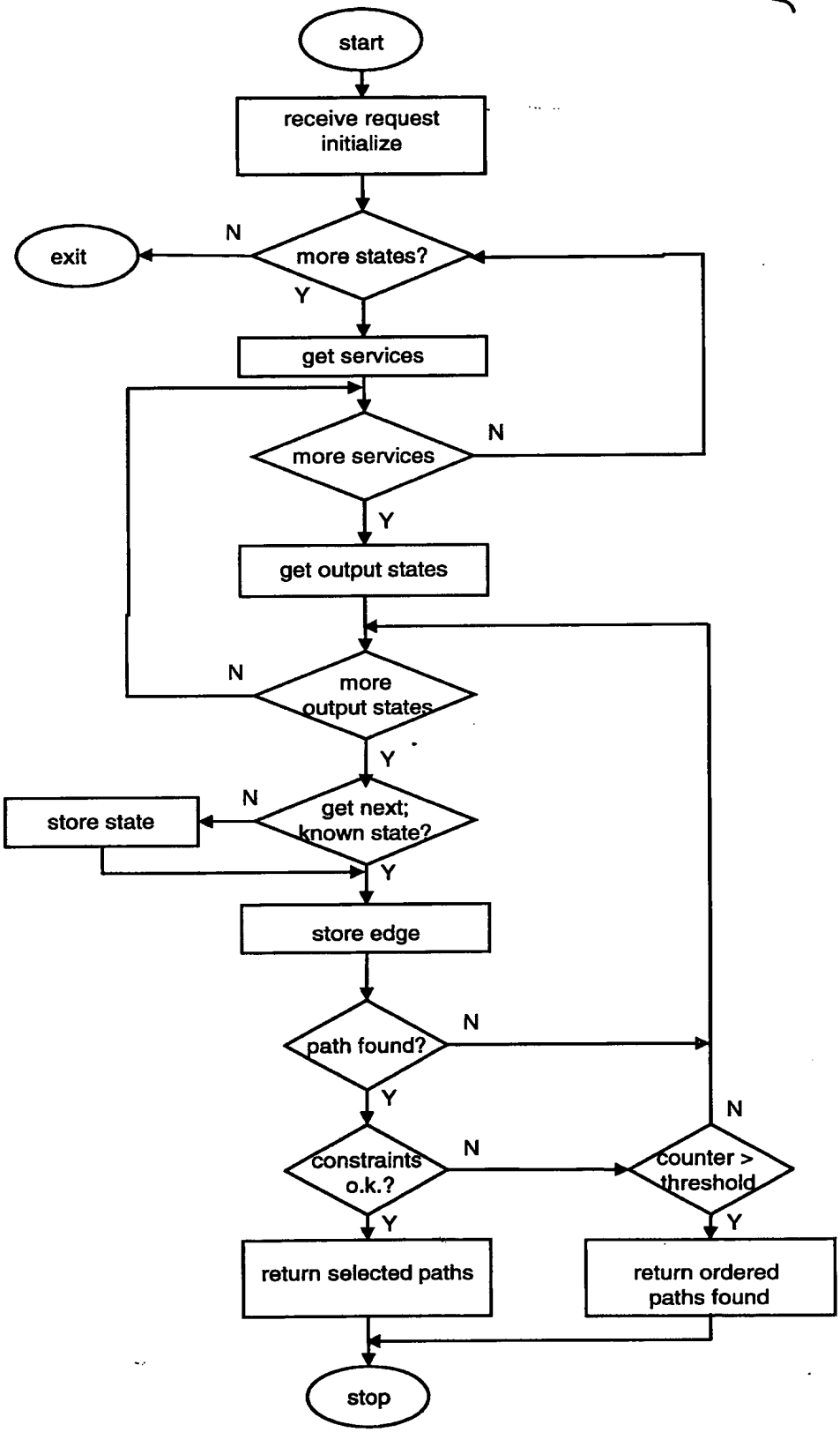

Fig.3

ID	as1
Type	print
PDL	Postscript
PPM	70
Color	Yes
Staple	No
Accounting	1
Location	3G76.3

ID	as2
Type	Conversion
Subtype	Page Coding
Input	PDF
Output	Postscript
Accounting	0 ct/unit
Location	1B

ID	as3
Type	Conversion
Subtype	Page Coding
Input	PCL
Output	Postscript
Accounting	1 ct/unit
Location	8E

ID	as4
Type	Conversion
Subtype	Page Coding
Input	PCL
Output	PDF
Accounting	0 ct/unit
Location	1B

ID	as5
Type	Conversion
Subtype	Page Coding
Input	EMF
Output	PCL; HPGL
Accounting	1 ct/unit; 5 ct/unit
Location	7B

ID:	as5;
Type:	Coding-Conversion;
Input:	PDL EMF;
Ouput:	PDL PCL, PDL HPGL;
Constraints:	accounting: price 1ct/Mbyte, location: lid nl5916pl.